BIL401 – Big Data – Final Report 17.12.2023

Berkay Yıldız 201104087
Zeynep Meriç Aşık 201410026

Abstract

This project delves into data analysis and machine learning using a dataset of 20,058 chess games from Lichess.org. Divided into two phases, the first involves data analysis to discern conditions leading to player victories, losses, or draws. The second phase employs machine learning to predict match outcomes based on relevant attributes. Using Apache Spark and other Big Data methods ensures efficient handling of the vast chess dataset. This project contributes to understanding chess dynamics, offering practical insights for strategic gameplay and also learning more on utilizing Apache Spark. Dataset can be obtained from https://www.kaggle.com/datasets/datasnaek/chess

I. Introduction

In this project, the problem that will be focused on is to correctly analyse the dataset and predict the winner based on the given game attributes. Chess, a strategic board game that has stood the test of time, serves as a captivating domain for data analysis and machine learning exploration. With the availability of vast datasets from platforms like Lichess.org, we have an opportunity to dig into the chess universe and uncover patterns that may influence game outcomes.

The primary objective of this project is twofold: firstly, to conduct a thorough data analysis by pre-processing and cleaning the dataset comprising 20,058 chess games. Through visualization and grouping techniques, we aim to differentiate key factors that contribute to a player's victory, defeat, or a draw. Understanding the conditions under which these outcomes occur can provide valuable insights into chess strategies and player behaviours.

In the second phase of the project, we transition to the realm of machine learning. Leveraging the pre-processed data, we aspire to develop a predictive model capable of forecasting the result of a chess match when presented with relevant attributes. This predictive capability not only serves as an interesting application of machine learning but also offers potential benefits for players seeking to enhance their strategic understanding and decision-making.

A pivotal aspect of this project involves the utilization of advanced data processing tools, particularly Apache Spark. Given the size of the dataset and the complexity of chess games, employing scalable and efficient technologies becomes imperative. It is also aimed that the project is able to successfully run over a bigger data than the one that is currently used and do not encounter any problems meanwhile. Spark, with its distributed computing capabilities, allows us to handle these data seamlessly, ensuring that our analyses and machine learning algorithms can scale to meet the demands of the task at hand.

By combining the richness of chess data, the power of data analysis, and the predictive capabilities of machine learning, this project endeavours to contribute to the broader understanding of chess dynamics. Whether uncovering hidden strategies, identifying influential game attributes, or predicting outcomes, the insights gained from this exploration

have the potential to enhance both the analytical and strategic facets of chess gameplay. As we navigate through the depths of the dataset, the process unveils valuable knowledge within the realm of chess analytics and machine learning.

II. Related Work

A. Data Analysis

In this phase, the initial step involves scrutinizing the data for cleanliness. A thorough inspection is conducted for null and duplicate values, revealing the absence of null values but the presence of some duplicates, which are subsequently removed. Subsequently, the dataset undergoes visualization. Analysis of the 'victory_status' indicates that games concluding with draws and those ending due to running out of time are notably fewer compared to resignations, which dominate, followed by checkmates. Further examination of the number of turns in all games reveals that draws and out-of-time scenarios tend to require more turns for game finalization.

Upon inspecting the target data, which comprises 'winner' information, an expected imbalance is observed. Draw values are notably lower compared to instances where either the black or white player emerges victorious. This imbalance poses challenges in predicting 'draw' values in machine learning models.

Moving forward, a meticulous exploration of the highest number of wins for both players unveil insights into the openings contributing the most to each player's success. The top 10 openings are enumerated for both white and black players. Notably, The Sicilian Defence emerges as a potent opening for both, with Van't Kruijs Opening proving influential for black players and Scandinavian Defense: Mieses-Kotroc Variation for white players. Interestingly, some of these openings are similar, indicating that the player initiating the game (typically the white player) does not significantly impact the type of opening employed.

The dataset also has the information of the ratings of players. To observe this data, the difference between the ratings is taken. This inspection showed that the difference of ratings of players are lower on the result ‘draw’ which make sense to why the match has no winners. However, black and white player seem to win on certain circumstances which is having a large gap between ratings. Subsequently, the impact of the rating difference on the number of turns is examined. The analysis reveals that as the rating difference between players decreases, the number of turns in the game tends to increase, aligning with expectations.

Following these insights, categorical data is transformed into numerical data through One-hot Encoding, and the distributions are presented as percentages. The pie charts highlight that 80% of the rows are rated, with resignations dominating the victory status, constituting more than half of the data. The distribution of 'winner' is approximately 50% for white, 45% for black, and nearly 5% for draws. Conversion of time increment, opening name, and ECO features into numerical data exposes a significant number of distinct labels, raising questions about their quality of providing information for data analysis or model training.

Subsequent decisions are made to reshape or drop certain columns before proceeding. Initial columns deemed insufficient in providing relevant information include start and end times. While the actual time is deemed unnecessary, the game duration is considered valuable. The time interval between start and end times is calculated and added as the 'game_duration' column, though precision issues with the time values may affect analytical accuracy.

Columns such as 'id', 'white_id', and 'black_id' are dropped, as the goal is to analyse moves and outcomes rather than specific player skills. Additionally, 'created_at' and 'last_move_at' columns are discarded, as their difference is incorporated into the new 'game_duration' column. Other columns in One-hot form ('_ind') are removed.

Anomalies were identified during the evaluation of turns, specifically cases where games concluded in the 0th turn, indicating potential data corruption. In our outlier detection methodology, we observed instances where the lower range attained nonsensical values. However, adjustments made to this lower range had minimal impact on those instances, thereby not significantly affecting the data quality. Consequently, a decision was made not to conduct separate outlier detection for each feature, as modifying the lower range, for example, for 'turns' resulted in a negative value, an impossible scenario in a chess game. Yet, even when adjusting the lower range to a small positive number, the outlier value remained unaffected.

Apart from this scenario, all other outliers were not removed as it was deemed inappropriate to do so. For instance, 'winner_ind' was a numerical table we generated ourselves, and the removal of rows related to 'victory_status_ind' for drawn games was not desired. This decision aimed to preserve the integrity of the dataset without compromising its relevance and usability.

Based on the correlation matrices provided in the 'tables' section (table-1, table-2, table-3), the derived meanings are as follows:

i. Strongly Positive Correlation between Black Ratings and White Ratings: This indicates that players with higher ratings tend to play more frequently with others having similar high ratings. This suggests a linkage between player skill levels, showing a preference for higher-rated players to engage with similarly skilled counterparts.

ii. Strongly Positive Correlation between Opening_Name_Ind and Opening_Eco_Ind: This demonstrates a robust relationship between opening moves and economic classifications. Certain opening moves or strategies are likely strongly associated with specific economic classifications, indicating a high probability of a particular economic classification when a specific opening move is employed.

iii. Strongly Positive Correlation of Opening_Ply with Opening_Eco_Ind and Opening_Name: This highlights a connection between the length of opening moves and either economic classifications or specific opening moves. The length of certain opening moves shows a strong relationship with particular economic classifications or opening moves. Longer opening sequences generally correspond to specific strategies or classifications.

iv. Positive Correlation between Turns and Game_Duration: This implies a relationship between the number of moves made during a game and the duration of the game. Longer games typically involve a greater number of moves. This correlation is quite natural, as longer games often necessitate more moves.

v. Stronger Positive Correlation between Black Ratings and White Ratings in Draw Scenarios: This suggests that if a game ends in a draw, the ratings of black and white players are more strongly correlated. In such instances, the relationship between player ratings might be more pronounced or robust when the game results in a draw.

These interpretations elucidate the correlations among various attributes, indicating their associations with different aspects of the game, such as strategies employed, move counts, and player performance.

vi. Research on Opening Names: Before deriving meanings from openings and understanding how machine learning could extract significance from them, tables were generated regarding the relevance of win rates based on opening names in the dataset. The goal was to find the most suitable opening names for white player wins, black player wins, and achieving a draw. Two methods were employed for this purpose.

- Opening Win-Draw Ratio Table:

This table looked into opening names that resulted in the most wins for both white and black players, alongside identifying openings that led to the highest number of draws. It was observed that rarely played and less probable scenarios were disproportionately highlighted. Due to the inadequacy in the number of occurrences in our dataset, opening names played and won only once stood out with a high win percentage, overshadowing others. This hindered drawing meaningful conclusions, prompting the decision to create a different table for interpretation. (table-6)

- Opening Win-Draw Scoring Table:

To mitigate the limitations observed earlier, a scoring table was devised. When the desired outcome occurred, the opening received a score of 1, while undesired outcomes garnered a -1 score. In cases where the game was played very few times, the aim was for the opening name to receive a low score. This methodology resulted in a distinct scoring table, providing an alternative perspective and was subsequently shared for analysis. (table-7)

From the Spark UI that can be super visioned from the link below SparkContext. All of the information about jobs can be viewed from this page such as environment information (table-8), summary (table-9), stages (table-10), event timeline (table-11), and all completed jobs (table-12).

The tables are just a partition of the actual analytics that are provided from SparkContext. Another addition to the given information about data analysis is that toPandas() method is used in some parts of the code for visualization purposes, they do not affect the computation of any results and can be deleted if a greater dataset would be used.

More Pre-Processing to Make the Data ML Ready

With a thorough analysis encompassing visualization and numerical conversion completed, the focus shifts towards preparing the data for training machine learning models. Various methods are explored, and the most effective ones are selected. Regarding scaling, given the data's unbalanced and skewed nature, the initial choice was the Robust Scaler. However, the utilization of the Standard Scaler yielded clearer results in terms of model training. Due to the limited amount of data available, a decision was made to utilize a train-test split ratio of 0.9 to 0.1.

Throughout the experimentation with these parameters and the training of models, a late realization surfaced regarding data leakage. In the 'victory_status' column, one of the labels is 'draw,' which is also a label in the target column 'winner.' This issue was identified when the trained model accurately predicted all draw values but achieved only around 60% accuracy for other outcomes. Consequently, the 'victory_status' column is dropped just before training, following a detailed analysis.

To further delve into the data, Principal Component Analysis (PCA) is applied. This reduces the data dimension from 9 to 3 for visualization purposes, enhancing the exploration of underlying patterns and relationships within the dataset.

Now, it is time to carry on with model training.

B. Machine Learning

Following the pre-processing steps and deriving new insights through analysis, several popular training models, including Linear Regression, Random Forest, Decision Tree, and Neural Network, are assessed on the dataset. However, the analytical outcomes are not particularly promising.

Linear Regression, surprisingly, outperformed the other models, achieving metrics almost on par with Neural Network at approximately 62% accuracy. The rationale behind this unexpected outcome will be elucidated shortly. Decision Tree and Random Forest produced closely aligned results, both hovering around 60% accuracy.

Numerous methods were explored in attempts to enhance metric scores. However, upon dropping the 'victory_status' column, the model struggled to accurately predict any 'draw' values. Improvements were not observed for cases where either the black or white player emerged victorious. To investigate further, the decision tree was examined after training which can be viewed as table-5, revealing that it predominantly relied on two features: white rating (feature 1) and black rating (feature 2). Consequently, the tree made decisions based solely on rating differentials, favouring the player with the higher rating. This singular focus on rating discrepancies explains the model's inability to predict 'draw' values, resulting in nearly half of the predicted draws being classified as white wins, with the remainder labelled as black wins as it can be seen from table-4.

III. Results

In summary, this case journeyed through an exploration of player ratings as pivotal factors, revealing that a reliance on only two features could not fulfil the initial project expectations. The modelling process fell short, especially in addressing the intricate scenario of games ending in draws, leaving this aspect largely unaccounted for.

It's acknowledged that player ratings are significant factors. However, we initially hypothesized that opening moves could also carry substantial weight and provide insights. The limited usability of this feature stemmed from both the scarcity of data and the abundance of various opening types. For instance, the maximum number of games played for the highest-rated openings was notably low: 228 for white, 54 for black, and merely 4 for draws. Considering we had over 20,000 data points, these numbers were insufficient for meaningful analysis. Another example highlighted that the most successful opening moves had been played in fewer than 15 games.

An intriguing revelation surfaced as the Linear Regression model unexpectedly outperformed others, even rivalling the Neural Network. This peculiarity might be rooted in the neural network's underperformance due to the limited data available for deep learning.

Consequently, the problem transitioned into a linearly solvable one, as the 'draw' parameter was omitted, reducing the dimensionality from 3 to 2 and focusing on deciding which rating is superior. This shift contributed to the Linear Regression model's marginally better performance.

In conclusion, extensive data analysis uncovered that the dataset lacks significant high-dimensional or relational aspects. Despite this, the opening names emerge as potentially informative, suggesting applications such as a website where users can input white or black player information to receive recommendations on opening styles. While the predictive capabilities may not have reached the envisioned heights, the insights gained pave the way for practical applications in the realm of chess strategy.

IV. TABLES

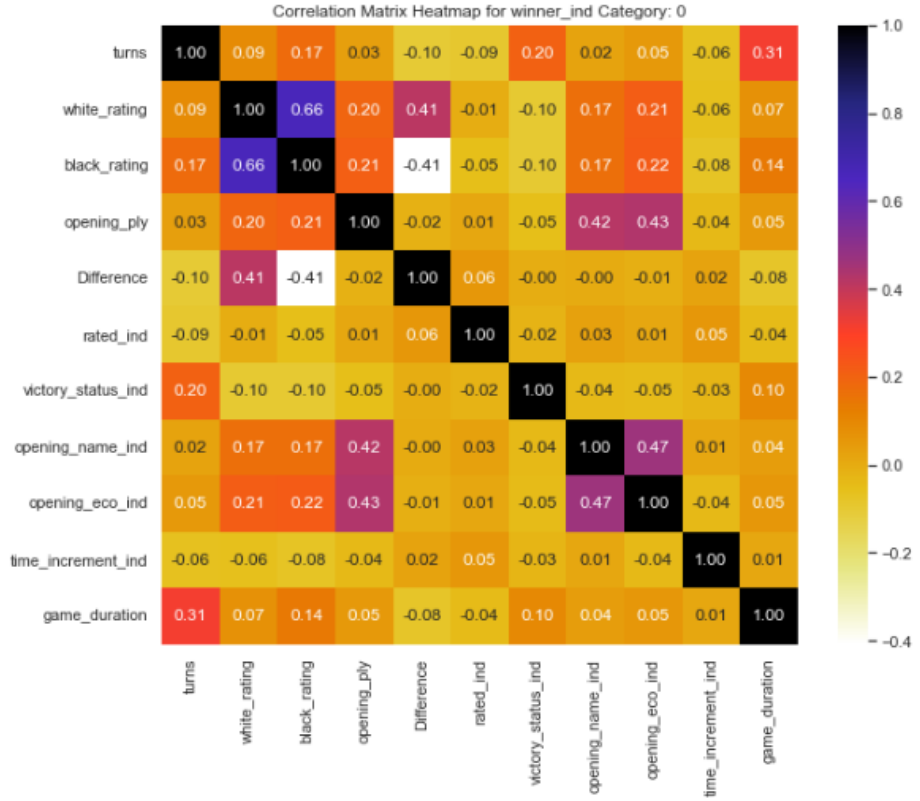

TABLE-1

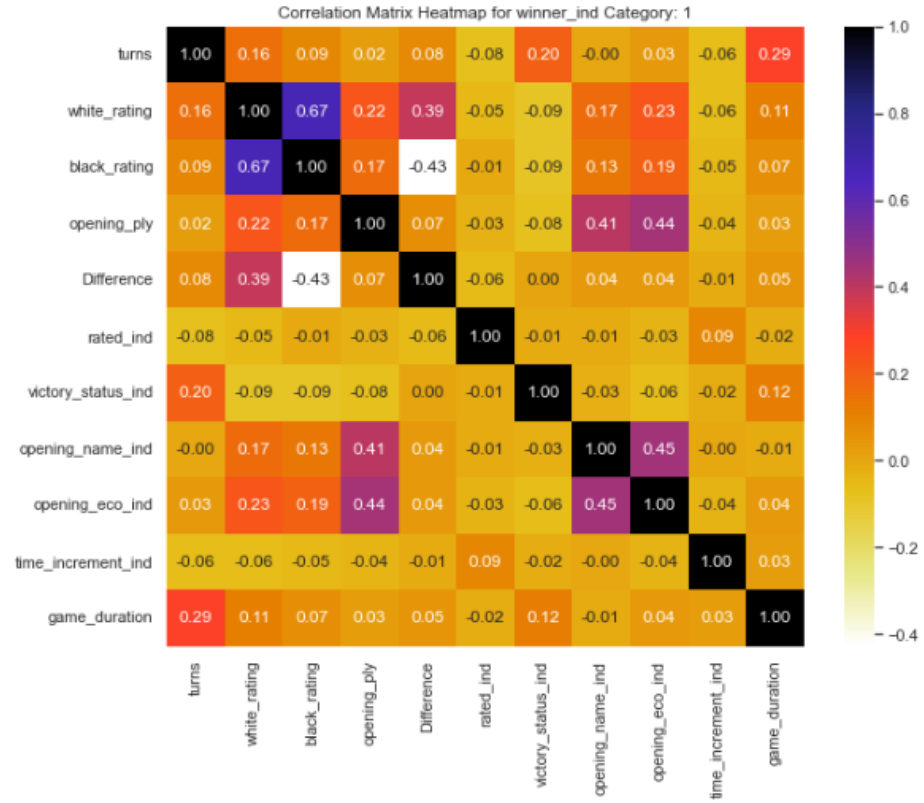

TABLE-2

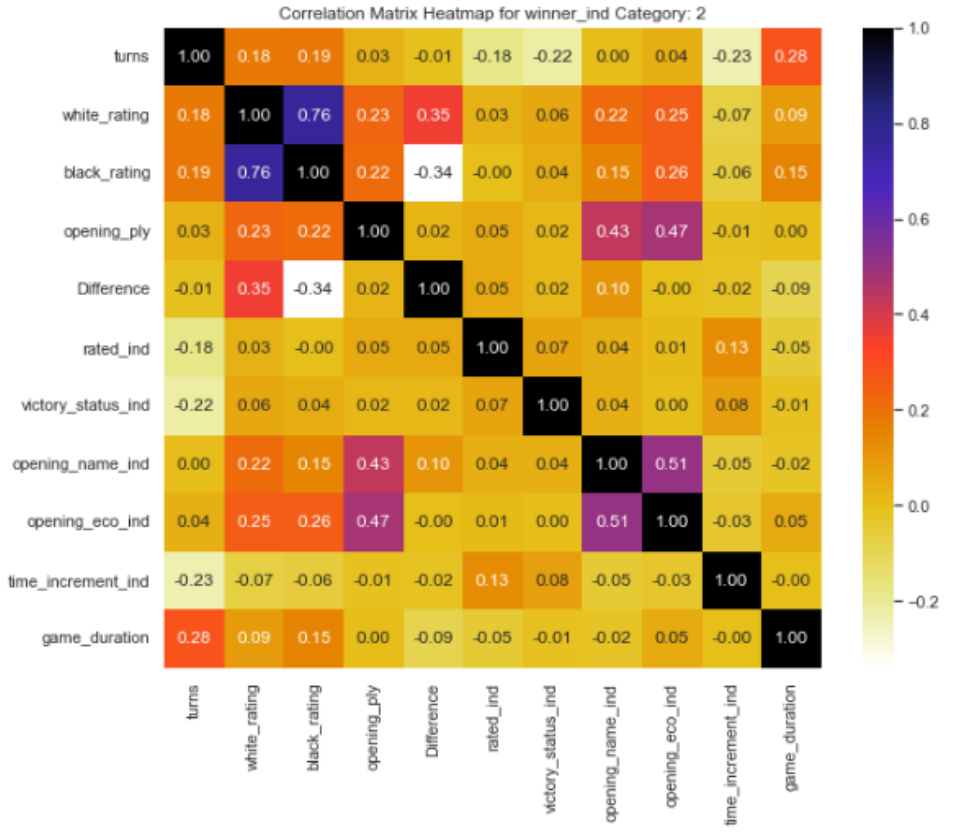

TABLE-3

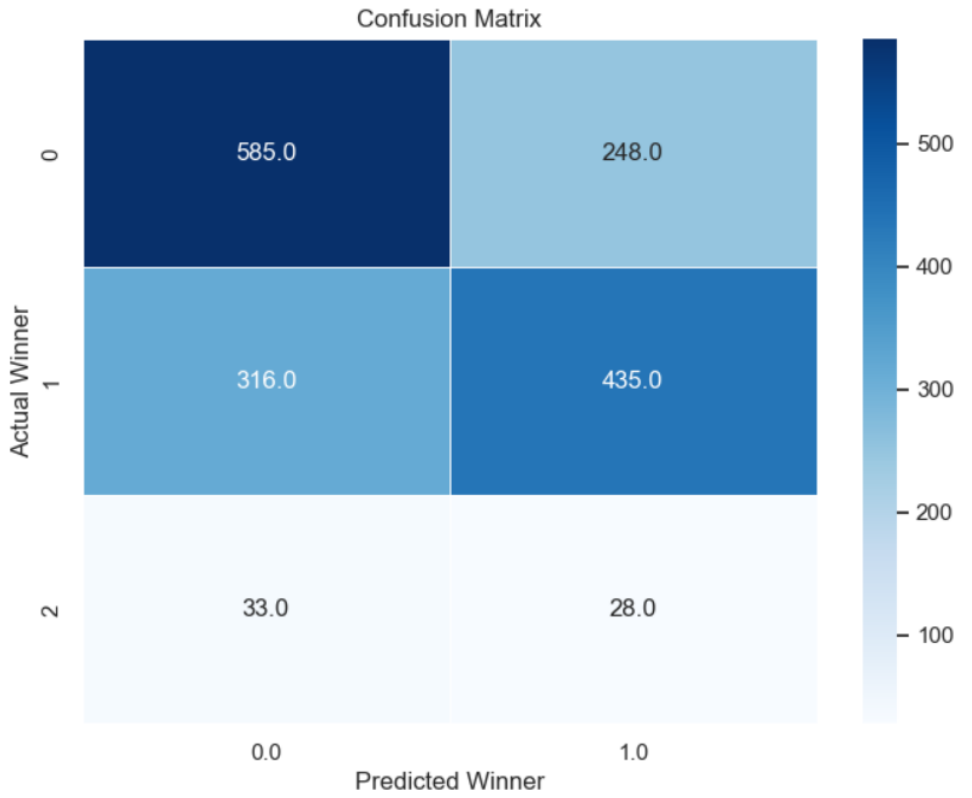

TABLE-4

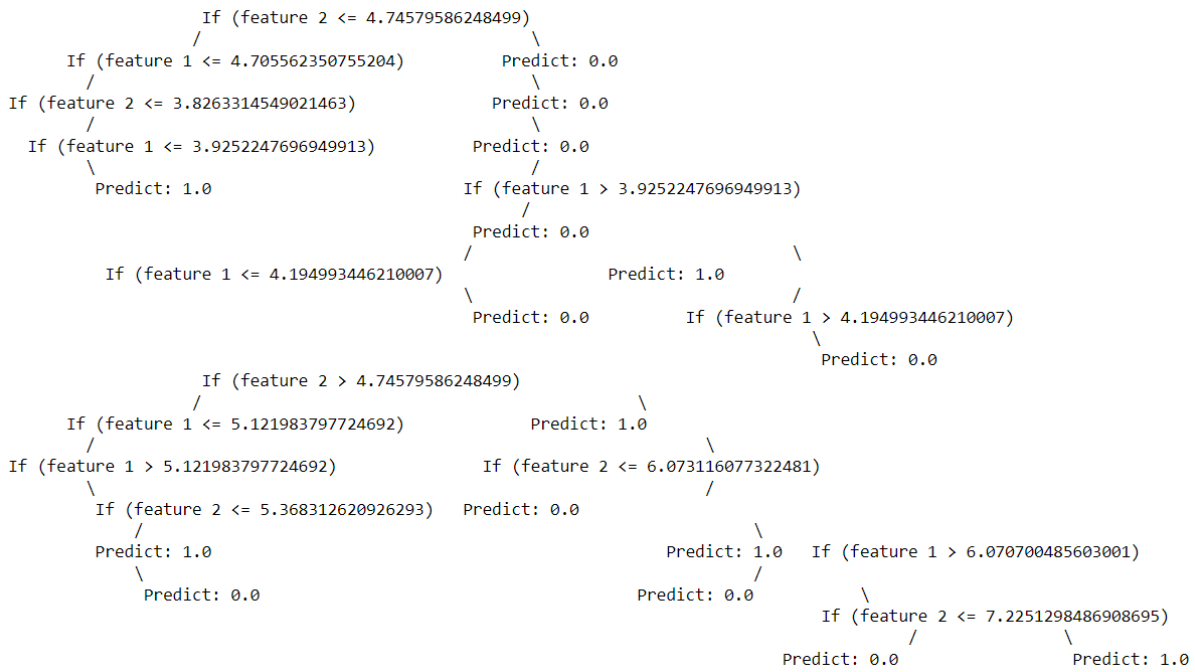

TABLE-5

Openings with the Highest White Player Winning Rate:

opening_name	total_count	white_wins_rate
Scotch Game: Classical Variation | Intermezzo Variation	14	100.0
Tarrasch Defense: Symmetrical Variation	9	100.0
Scotch Game: Haxo Gambit	9	100.0
Queen's Gambit Declined: Harrwitz Attack	9	100.0
Dutch Defense: Normal Variation	9	100.0

Openings with the Highest Black Player Winning Rate:

opening_name	total_count	black_wins_rate
Indian Game: Anti-Gruenfeld | Advance Variation	4	100.0
King's Pawn Game: Alapin Opening	3	100.0
Four Knights Game: Spanish Variation | Rubinstein Variation Accepted	2	100.0
Caro-Kann Defense: Modern Variation	2	100.0
Sicilian Defense: Wing Gambit | Deferred Variation	2	100.0

Openings with the Highest Draw Ending Rate:

opening_name	total_count	draws_rate
Sicilian Defense: Kupreichik Variation	1	100.0
Old Indian: Czech Variation	1	100.0
Nimzo-Indian Defense: Reshevsky Variation	1	100.0
French Defense: Baeuerle Gambit	1	100.0
French Defense: Tarrasch Variation | Pawn Center Variation	1	100.0

TABLE-6

```
Openings with Highest Scores (for White Player):
+--------------------------------+-----------+-----+
|opening_name                    |total_count|score|
+--------------------------------+-----------+-----+
|Scotch Game                     |228        |72   |
|Queen's Pawn Game: Mason Attack |196        |57   |
|French Defense: Knight Variation|242        |57   |
|Sicilian Defense                |311        |55   |
|Sicilian Defense: Bowdler Attack|257        |46   |
+--------------------------------+-----------+-----+
```

```
Openings with Highest Scores (for Black Player):
+-----------------------------------+-----------+-----+
|opening_name                       |total_count|score|
+-----------------------------------+-----------+-----+
|King's Pawn Game: Napoleon Attack  |54         |8    |
|Vienna Game                        |27         |7    |
|Four Knights Game                  |31         |6    |
|King's Pawn Game: Nimzowitsch Defense|27       |5    |
|Van Geet Opening                   |30         |5    |
+-----------------------------------+-----------+-----+
```

```
Openings with Highest Scores (for Draw):
+-------------------------------------------------------------+-----------+-----+
|opening_name                                                 |total_count|score|
+-------------------------------------------------------------+-----------+-----+
|Hungarian Opening: Sicilian Invitation                       |4          |2    |
|Nimzo-Indian Defense: Reshevsky Variation                    |1          |1    |
|Italian Game: Evans Gambit |  Stone-Ware Variation           |1          |1    |
|French Defense: Tarrasch Variation |  Pawn Center Variation  |1          |1    |
|Old Indian: Czech Variation                                  |1          |1    |
+-------------------------------------------------------------+-----------+-----+
```

TABLE-7

Environment

▾ **Runtime Information**

Name	Value
Java Home	C:\Program Files\Java\jdk-19
Java Version	19 (Oracle Corporation)
Scala Version	version 2.12.18

▾ **Spark Properties**

Name	Value
spark.app.id	local-1702038673212
spark.app.name	pyspark-shell
spark.app.startTime	1702038671934
spark.app.submitTime	1702038671785

TABLE-8

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(1)	0	252 KiB / 434.4 MiB	0.0 B	8	0	0	242	242	13 min (0.5 s)	605 MiB	61.9 MiB	47.5 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	252 KiB / 434.4 MiB	0.0 B	8	0	0	242	242	13 min (0.5 s)	605 MiB	61.9 MiB	47.5 MiB	0

TABLE-9

Completed Stages (223)

Page: 1 2 3 > 3 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
401	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:14	0,9 s	1/1			26.3 KiB	
398	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:13	0,3 s	1/1			1432.9 KiB	26.3 KiB
396	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:13	0,3 s	1/1	7.3 MiB			1432.9 KiB
395	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:12	0,9 s	1/1			27.5 KiB	
392	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:11	0,3 s	1/1			1432.9 KiB	27.5 KiB
390	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:11	0,4 s	1/1	7.3 MiB			1432.9 KiB
389	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:10	0,9 s	1/1			18.7 KiB	
385	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:10	0,2 s	1/1			28.0 KiB	18.7 KiB
382	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:10	40 ms	1/1			28.0 KiB	
379	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:09	0,4 s	1/1			1432.9 KiB	28.0 KiB
377	showString at NativeMethodAccessorImpl.java:0 +details	2023/12/08 15:35:09	0,4 s	1/1	7.3 MiB			1432.9 KiB
376	toPandas at C:\Users\Asus\AppData\Local\Temp\ipykernel_19440\1568979172.py:1 +details	2023/12/08 15:33:10	0,1 s	2/2			1964.2 KiB	
374	toPandas at C:\Users\Asus\AppData\Local\Temp\ipykernel_19440\1568979172.py:1 +details	2023/12/08 15:33:09	0,3 s	1/1	7.3 MiB			1964.2 KiB
373	collect at C:\Users\Asus\AppData\Local\Temp\ipykernel_19440\1766347761.py:4 +details	2023/12/08 15:33:06	0,1 s	2/2			1964.2 KiB	
371	collect at C:\Users\Asus\AppData\Local\Temp\ipykernel_19440\1766347761.py:4 +details	2023/12/08 15:33:06	0,3 s	1/1	7.3 MiB			1964.2 KiB

TABLE-10

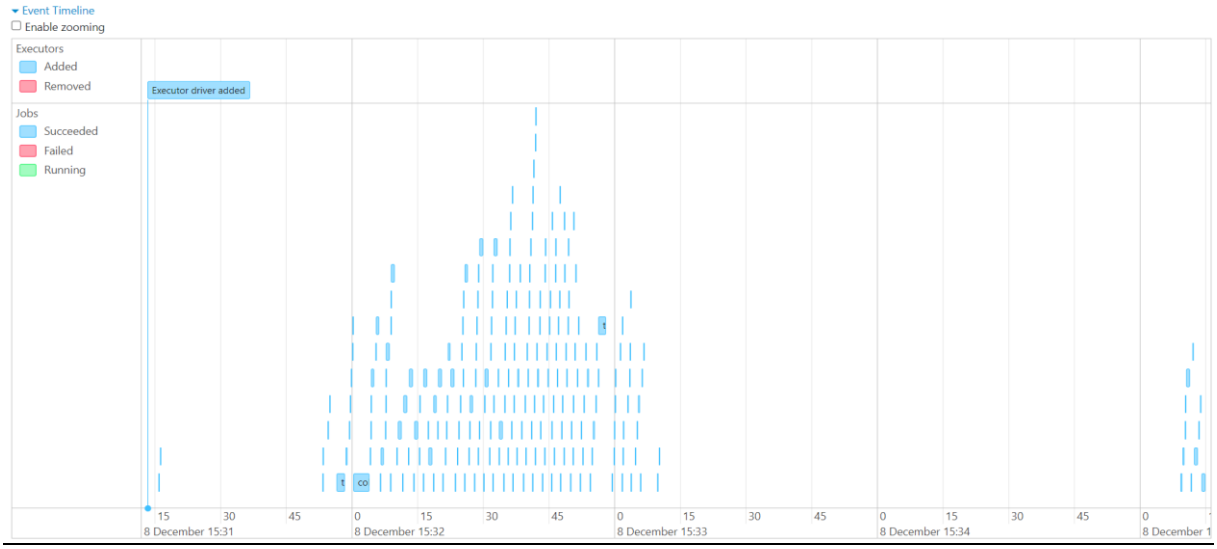

TABLE-11

▾ **Completed Jobs (223)**

Page: 1 2 3 > 3 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
222	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:14	0,9 s	1/1 (2 skipped)	1/1 (2 skipped)
221	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:13	0,3 s	1/1 (1 skipped)	1/1 (1 skipped)
220	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:13	0,3 s	1/1	1/1
219	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:12	0,9 s	1/1 (2 skipped)	1/1 (2 skipped)
218	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:11	0,3 s	1/1 (1 skipped)	1/1 (1 skipped)
217	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:11	0,4 s	1/1	1/1
216	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:10	0,9 s	1/1 (3 skipped)	1/1 (3 skipped)
215	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:10	0,2 s	1/1 (2 skipped)	1/1 (2 skipped)
214	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:10	50 ms	1/1 (2 skipped)	1/1 (2 skipped)
213	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:09	0,5 s	1/1 (1 skipped)	1/1 (1 skipped)
212	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/12/08 15:35:09	0,4 s	1/1	1/1

TABLE-12